FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

A Framework for Mixed-Reality Simulations of Smart-Spaces

Flávio Henrique Ferreira Couto

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Hugo José Sereno Lopes Ferreira

Co-Supervisor: André Monteiro de Oliveira Restivo

July 29, 2018

A Framework for Mixed-Reality Simulations of Smart-Spaces

Flávio Henrique Ferreira Couto

Mestrado Integrado em Engenharia Informática e Computação

July 29, 2018

Abstract

The Internet of Things (IoT) is a recent trend that aims to connect the physical and digital world even more. The increase in the number of devices connected to the Internet does not show signs to stop, covering diverse spaces from homes to industries. In order to guarantee the correct functionality of said devices, proper testing is needed. Despite the rich testing environment of other areas of software engineering, IoT solutions present new challenges to developers, such as tight coupling between hardware and software, the interaction between devices, real-time data transfers, and network issues.

To overcome those problems, some testing methods were developed, like pilot testing (providing a prototype of the solution to a small number of users), simulation (abstracting physical components from their software) and, more recently, formal verification (logical proof of functionality). Despite their corresponding advantages, scalability and interaction with the real world are difficult to properly be verified and the availability of testing frameworks is scarce and mostly centered around "Virtual Labs" and network simulators, where the hardware components and the data produced from their interaction with a physical environment are not able to be tested.

The goal of this project is to develop an event-based simulation which provides a testing framework with the ability to monitor communication between the devices, control behaviors and generate usage data. These features allow developers to execute performance testing, AI training and interoperability testing without needing to implement a physical prototype of the solution. In order to bridge the gap between the simulator and the real world, we propose an approach we coin as “mixed-reality”, in which the devices in the simulation can be mapped to physical devices, enabling the testing of modifications to already implemented solutions and the gathering of real-world data.

The developed system is tested against use case scenarios of IoT environments, fully or partially simulated, showcasing the features of the said framework.

Resumo

Internet of Things (IoT) é uma tendência recente que visa conectar ainda mais os mundos físico e digital. O aumento no número de dispositivos conectados à Internet não demonstra sinais de abrandar, abrangindo diversos espaços desde casas a indústrias. Para garantir o correto funcionamneto dos referidos dispositivos, são necessários testes adequados. Apesar do rico ambiente de teste de outras áreas de engenharia de software, as soluções IoT apresentam novos desafios aos desenvolvedores, como o acoplamento apertado entre hardware e software, a interação entre dispositivos, transferências de dados em tempo real e problemas de rede.

Para superar esses problemas, foram desenvolvidos alguns métodos de teste, como testes piloto (fornecendo um protótipo da solução para um pequeno número de usuários), simulação (abstraindo os componentes físicos do seu software) e, mais recentemente, verificação formal (prova lógica da sua funcionalidade). Apesar das vantagens correspondentes, a escalabilidade e a interação com o mundo real são difíceis de verificar corretamente e a disponibilidade de sistemas de teste é escassa e principalmente centrada em ”laboratórios virtuais” e simuladores de rede, onde os componentes de hardware e os dados produzidos pela sua interação com o ambiente físico não podem ser testado.

O objetivo deste projeto é desenvolver uma simulação baseada em eventos que forneça um sistema de testes com a capacidade de monitorizar a comunicação entre os dispositivos, controlar comportamentos e gerar dados de uso. Esses recursos permitem aos desenvolvedores executar testes de desempenho, treinos de sistemas de Inteligência Artificial e testes de interoperabilidade sem necessitar de implementar um protótipo físico da solução. A fim de transpor a barreira entre o simulador e o mundo real, propõe-se uma abordagem denominada como ”mixed-reality”, na qual os dispositivos na simulação podem ser mapeados para dispositivos físicos, permitindo o teste de modificações a soluções IoT já implementadas e recolha de dados do mundo real.

O sistema desenvolvido é testado através de casos de uso de ambientes IoT, total ou parcialmente simulados, nos quais as funcionalidades deste sistema serão demonstradas.

Acknowledgements

Firstly, I would like to thank my supervisor, Hugo Sereno Ferreira, for guiding me in the process of defining and creating a novel contribution to the IoT testing environment.

Secondly, I thank my second supervisor, André Restivo, for providing new perspectives for the developed system.

As important as ever, I give the biggest of gratitudes to my family, especially my parents, for supporting me all these years in fulfilling my desire to follow this path.

Lastly, but certainly not least, a special thanks to all my friends. It was with them that I've striven all these years to achieve this moment.

Flávio Henrique Ferreira Couto

"As the Internet of things advances, the very notion of a clear dividing line between reality and virtual reality becomes blurred, sometimes in creative ways."

Geoff Mulgan

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Motivation and Objectives . . . 2
- 1.3 How to read this dissertation . . . 2

2 Background **3**
- 2.1 Testing Levels and Methods . . . 3
- 2.2 Testing the Internet-of-Things . . . 4
- 2.3 Discrete-Event Simulation . . . 5
- 2.4 Conclusion . . . 7

3 State of the Art **9**
- 3.1 Internet-of-Things Testing Solutions . . . 9
- 3.2 IoT Testing Solutions Comparison . . . 12
- 3.3 Conclusion . . . 13

4 Thesis Statement **15**
- 4.1 Current Issues . . . 15
- 4.2 Main Goal . . . 15
  - 4.2.1 Use Case 1: prototype an IoT environment without any physical device . . . 16
  - 4.2.2 Use Case 2: use physical sensors in a simulated environment . . . 16
  - 4.2.3 Use Case 3: provide simulated sensor data to the physical environment . . . 16
  - 4.2.4 Use Case 4: control physical actuators from a simulated environment . . . 16
  - 4.2.5 Use Case 5: Execute test batteries . . . 16
- 4.3 Research Questions . . . 16
- 4.4 Evaluation Strategy . . . 17
- 4.5 Conclusions & Expected Contributions . . . 17

5 Implementation **19**
- 5.1 Solution Overview . . . 19
- 5.2 Components Overview . . . 20
  - 5.2.1 Simulation Framework . . . 20
  - 5.2.2 MQTT Broker . . . 24
  - 5.2.3 IoT Device . . . 27
- 5.3 Achieving Mixed-Reality Simulation . . . 27
  - 5.3.1 Simulated Devices . . . 27
  - 5.3.2 Physical Devices . . . 29
- 5.4 Conclusions . . . 31

6 Evaluation **33**
6.1 Use Case Scenarios . . . 33
6.1.1 Use Case Scenarios Definition . . . 33
6.1.2 Scenario 1: virtual environment . . . 34
6.1.3 Scenario 2: linking a physical switch to a physical light . . . 34
6.1.4 Scenario 3: physical light updating own state based on simulated switch status . . . 35
6.1.5 Scenario 4: testing a 24-hour life cycle of self-controlled light based on a luminosity sensor at the simulation rate of 1h/min . . . 35
6.2 Conclusions . . . 36

7 Conclusions and Future Work **37**
7.1 Main Problems . . . 37
7.2 Conclusions . . . 37
7.3 Future Work . . . 38

References **39**

A Use Case Scenarios Data **41**
A.1 Scenario 1: virtual environment . . . 41
A.2 Scenario 2: linking a physical switch to a physical light . . . 45
A.3 Scenario 3: physical light updating own state based on simulated switch status . . . 49
A.4 Scenario 4: testing a 24h lifecycle of self-controlled light based on a luminosity sensor at the rate of 1h/min . . . 53

List of Figures

2.1 Typical layer composition of an IoT system. 4
2.2 Discrete event simulation execution flowchart 6

5.1 Architectural components of the IoT simulation system 19
5.2 Modified event simulation execution flowchart 23
5.3 Class diagram for the simulation framework 24
5.4 Property Value Change Event on a Simulated Device 28
5.5 Property Value Change Request Event on a Simulated Device 28
5.6 Property Value Change Event on a Physical Device 29
5.7 Property Value Change Request Event on a Physical Device 29

LIST OF FIGURES

List of Tables

3.1 Overview comparison of the available tools on the IoT testing landscape 14

Abbreviations

API	Application Programming Interface
IoT	Internet of Things
MQTT	Message Queuing Telemetry Transport
SUT	System Under Testing

Chapter 1

Introduction

The Internet of Things (IoT) is one of the more recent areas of software development. As any emergent trend, more and more IoT solutions were developed and deployed in order to showcase the new opportunities in this domain. However, with this new domain came new problems that must be properly tackled so that these systems are not only produced at a proper pace, but also with proven correctness.

1.1 Context

The Internet of Things can be described as having the objective to connect the world even more. It is a new paradigm in which everyday objects, such as a doorbell or the lights of a house, become part of the Internet, providing the ability to be monitored and interacted with, and bringing the physical and digital world together[CE11].

The growth of devices connected to the Internet has been apparent through recent years and many predictions have been produced. According to Gartner, over 20 billion *things* will be connected by the year 2020, with expected business expenses of almost $3 trillion [Gar17]. Such magnitude of devices brings with it several challenges on par with its size. Li et al. [LXZ15] present a list with some of the still open-challenges, namely, technical challenges (lack of reference architectures and protocols, heterogeneity and insufficient automation), lack of standardization, security and privacy issues, faulty development strategies, interoperability and support deficiencies.

Testing and validation of IoT systems are part of those challenges. Although an individual device provides few functionalities and, as such, is easy to properly be tested, the scale of complex IoT systems, with hundreds of devices deployed in real-world environments (labeled *smart spaces*), the interaction between devices and the unreliable connectivity make these systems as complex to test and validate as their size [TM17].

However, most of these research challenges have been addressed already in other fields like embedded computing, distributed computing, cloud computing, mission-critical or process automation systems. As such, knowledge from these research fields can be leveraged into the IoT scenario, improving the reliability and robustness of the IoT-based solutions.

1.2 Motivation and Objectives

With the barrier between digital and physical world becoming even thinner and the incremental ability of decision making of computers, people who are not so knowledgeable about technologies foment some distrust in embracing the Internet of Things, mostly due to concerns on their reliability and their security. As such, developers of the IoT must take this into account and assure the correctness and security of their developed systems. In order to do so, proper testing frameworks must be available, providing a reliable environment for IoT solutions to be developed.

After analyzing the current testing environment of IoT and finding its gaps and challenges, the main goal of this dissertation is set as developing a simulation framework able to conciliate the simulated devices with IoT devices located in the real-world, allowing them to interact with each other.

1.3 How to read this dissertation

The remaining sections of this dissertation observe the following structure:

- Chapter 2, "Background" (p. 3), describes key concepts about testing in general, and in testing in IoT in particular.
- Chapter 3, "State of the Art" (p. 9), describes the abilities of some testing frameworks, their implementation and their coverage of IoT testing.
- Chapter 4, "Thesis Statement" (p. 15), provides an overview of the problems to be tackled and presents a basis of requirements and research questions that must be answered by the proposed solution.
- Chapter 5, "Implementation" (p. 19), describes the implementation of the solution, from its components to its communication protocols.
- Chapter 6, "Evaluation" (p. 33), showcases the results of the evaluation of the developed system.
- Chapter 7, "Conclusions and Future Work" (p. 37), presents the results of the developed framework, pondering on its main issues and future objectives.

Chapter 2

Background

Testing is the process of identifying failures, where a failure consists of any variance between actual and expected results.

The Internet of Things relies on a combination of hardware, software, and architectures that enable real-world objects to sense and interact with the surrounding environment while being Internet-connected and uniquely identifiable [WADX15]. As such, in order to guarantee IoT-based system's performance, scalability, reliability, and, further, security, it is needed focus on testing the different layers and components that make part of the system from low-level/hardware specifications to high-level components. It is hard to draw a line between the low-level and high-level components in IoT-scope since they are strongly connected and dependent, however, the methods and techniques used for testing these are, typically, similar.

As such, different characterizations of tests are provided in the following sections, along with their objectives and their requirements.

2.1 Testing Levels and Methods

Testing approaches can be in one or more levels, depending on the scope of the test and objective. So, different test levels are defined, as follows [Bei03]:

Unit Testing Testing of individual hardware or software units or groups of related units [IEE90]. It consists of isolating each part of the system and showing that individual parts fit its requirements and functionalities.

Integration Testing Software and/or hardware components are combined and tested to check the interaction between them and how they perform together [IEE90].

System Testing Testing a complete, integrated system to check the system's compliance and behavior within the specified requirements [IEE90]

Acceptance Testing Formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable a customer, a user, or other authorized entity to determine whether or not to accept the system [IEE90]

Different methods can be used to test the *system under test* (SUT), namely, white-box testing [Ost02], gray-box testing [LJX+04] and black-box testing [Edw01]. These methods are hereby described:

White-box Testing The internals of the SUT are all visible and known, and, as such, this information can be used to create test scenarios. Additionally, white-box testing is not restricted to failure detection but is also able to detect errors.

Black-box Testing The SUT internal content is hidden and only knowledge about the system's or module's inputs and outputs is available, being closer to real-world use situations.

Gray-box Testing A mix of the two previous techniques is used. Information about the internals of the SUT is used, but, however, tests are conducted under realistic conditions, where only failures are detected.

2.2 Testing the Internet-of-Things

IoT systems are complex by nature, depending on different software and hardware components, modules and architectures, produced by many manufacturers and with different working properties. As such, diverse needs of testing appear in result of the different variables that need to be tested. One can identify various challenges, as, for example, the high-heterogeneity, large-scale, dynamic environment, real-time needs, security and privacy implications and the difficulty on test automation. Hence, different testing needs appear from the different IoT layers (Figure 2.1):

Edge Testing : Concerns about testing the more low-level parts of IoT system's, like microcontrollers (e.g. Arduino) and programmable logic controllers (PLC). Testing approaches like embedded system testing can be typically used to perform tests on the edge layer, asserting the edge devices against their specification [Koo11].

Fog Testing : Tests regarding the middle-point layer on IoT system's, normally composed of gateways. Software testing approaches can be seamlessly applied since the devices that belong

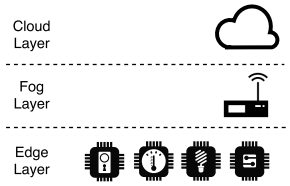

Figure 2.1: Typical layer composition of an IoT system.

to this layer have, typically, a comfortable amount of computing power and memory, running full operative systems (e.g. Linux). Additionally, since this is the connectivity-enabler layer, connecting the restraint devices and the Internet *per se*, it should cover network testing [KK16] and security testing [ZCW+14].

Cloud Testing : Cloud testing addresses the need to test the unique quality concerns of the cloud infrastructure such as massive scalability and dynamic configuration. This field has open-challenges and issues of its own, and they are extensible analyzed in the literature [BLC+11, RTS10].

To be able to test IoT systems as a whole, work has been pursued towards *IoT testbeds*, that enable to test the IoT systems from lower layers until the high-level ones. Although almost every testbed vertically encompasses all the layers, they are *single-domain*, focusing on a specific domain of application or technological aspect. Although there are some *multidomain* testbeds that combine different technologies into a common experimental facility. A survey on the currently active and publicly available physical testbeds is given by Gluhak et. al [GKN+11].

Along with physical testbeds another approach that has been pursued for testing IoT-based system's is the use of emulators and simulators. On one hand, an emulator is a system that behaves exactly like the target system (e.g. physical devices emulation). On the other hand, simulators enable a close replication of the target system but implemented in an entirely different way (e.g. smart city simulation). The work pursued by Looga et. al surveys the existent simulators and emulators, revealing issues on their suitability for testing IoT-based system and proposing a new emulation platform for the IoT, labeled MAMMotH [LODYJ12].

2.3 Discrete-Event Simulation

Discrete-Event Simulation is a simulation approach in which the behavior and performance of a complex system are portrayed as an ordered sequence of events [ZKP00]. IoT environments, being as complex as needed for their purpose, can and have been simulated using this approach.

In a discrete event simulation, each event describes a change in the simulated system at a specific time. Between each event, no changes in the system are expected to occur. This allows simulation systems to skip the time between events. Though more complex engines can be created according to dependencies between events, the bare-bones of the execution flow of this simulation approach is the one portrayed in Figure 2.2.

The first step of the simulation consists in inserting the initial events in the event queue in order to ensure that the simulation is able to start.

Then the system enters a loop in which every step the simulator retrieves the next event and processes it. The events are queued strictly in timestamp order to maintain causality and ensure that they do not affect earlier events.

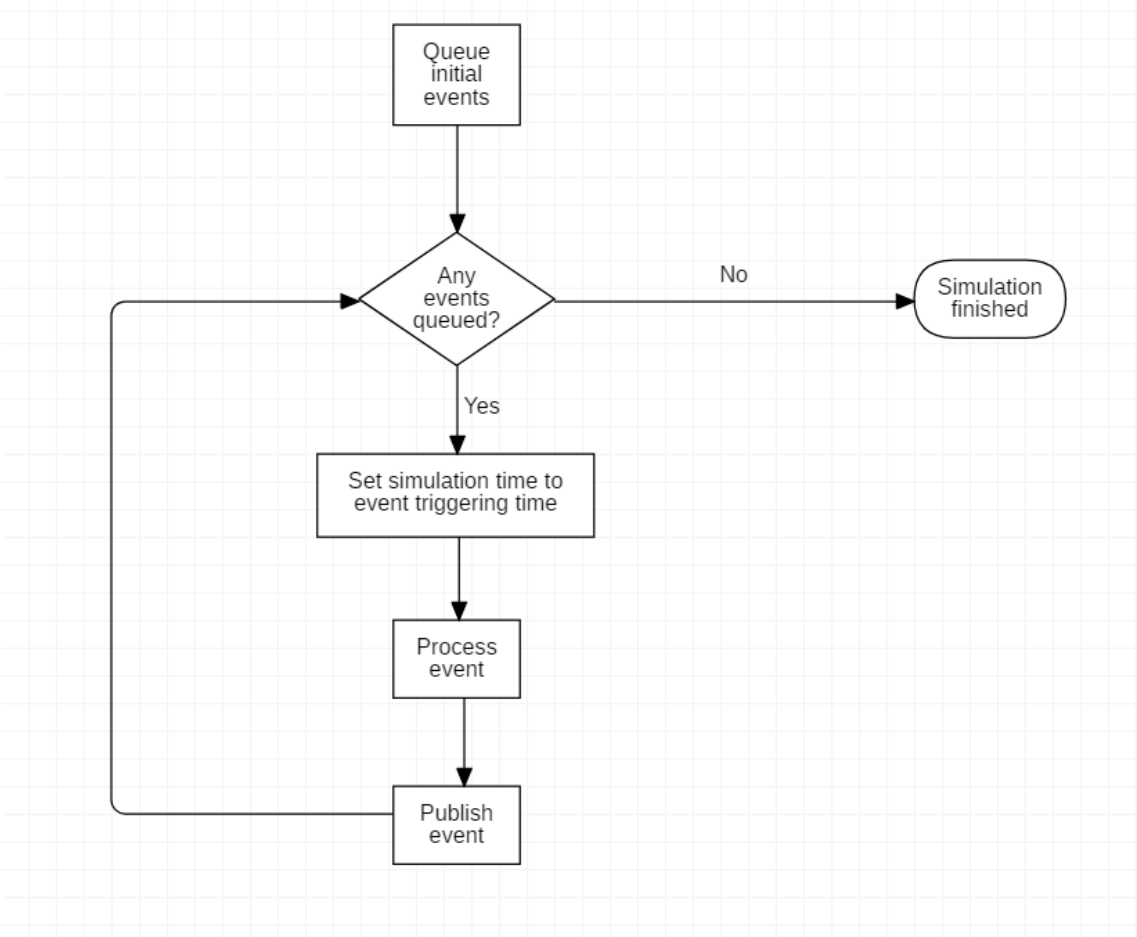

Figure 2.2: Discrete event simulation execution flowchart

Before processing and event, the simulation clock is set to the timestamp of the event, effectively advancing the system time. While processing the event, new events may be queued or previously queued events may be removed.

The simulation comes to an end when no more events are queued, or when no more simulation steps are deemed necessary. In this last step, statistical reports are usually generated and recorded.

2.4 Conclusion

This chapter presented key concepts of testing software systems in general and specific concepts for the IoT environments. As a popular technique for testing IoT systems and in which this dissertation solution is based upon, discrete-event simulation is also described. The next chapter will provide an overview of the state-of-the-art environments for testing IoT systems.

Chapter 3

State of the Art

3.1 Internet-of-Things Testing Solutions

As of today, there are already some solutions available for testing IoT-based systems. These solutions focus on different IoT layers and enabling technologies. An overview comparison and discussion of the tools available is given in Section 3.2. The tools have been selected after a curated search on scholar database (Scopus) and Google search engine, an are shortly described in the following paragraphs. The keywords used for the search were one or a combination of the following: `Internet-of-Things`, `test`, `testing` and `IoT`.

PlatformIO `http://platformio.org/`

PlatformIO is a cross-platform code builder and library manager, supporting nearly 200 development boards and most major embedded software development platforms. In spite of providing an Integrated Development Environment and Cloud Platform, PlatformIO's core is a console application built in Python that can be easily integrated with most popular code editors (e.g. Atom).

IoTIFY `https://iotify.io/`

IoTIFY is an application development environment for IoT without hardware dependencies. By resorting to device virtualization, it provides a virtual lab for building embedded prototypes and a network simulation for system scaling and data generation.

FIT IoT-LAB `https://www.iot-lab.info/`[ABF+15]

IoT-LAB is a scientific testbed for testing small wireless sensor devices and heterogeneous communicating objects built on a very large scale infrastructure, deployed around six sites in France with over 2000 sensor nodes. It is the successor of SENSLAB testbed and is part of the Future Internet of the Things (FIT) platform.

ArduinoUnit https://github.com/mmurdoch/arduinounit

ArduinoUnit is a unit testing framework for Arduino libraries. Being a lightweight library, developers can easily test their systems in an Arduino board, despite their low amount of resources. However, it is up to the developer to upload the testing application to the target board and the results must be interpreted by them, commonly through the use of a serial port monitor.

MAMMotH [LODYJ12]

MAMMotH is a large-scale IoT emulator, being able to emulate ten thousand devices per Virtual Machine, whose architecture presumes three distinct scenarios, namely: mobile devices connected via GPRS to a base station forming a star topology, a stand-alone wireless sensor network (WSN) connected to a base station via GPRS and constrained devices (e.g. sensors) connect to proxies, which in turn connect to the backend, a large-scale IoT emulator. In order to reproduce the communication problems present in a real IoT environment, the proxy to which the devices are connected simulates a radio link for each node, able to delay and drop messages. Developers can then use this setup to create experiment scenarios, deploy them on a testbed and monitor the results.

SimIoT [SBAM14]

SimIoT is a toolkit to achieve experimentation on dynamic and real-time multi-user submissions within an IoT scenario. The toolkit is based on the SimIC, a system that allows modelers to configure a diversity of clouds in terms of datacenter hosts and software policies wherein the desired number of users could send single or multiple requests for computational power, software resources, and duration of VM virtualization.

Cooja Simulator https://anrg.usc.edu/contiki/ [BE15]

The Cooja Simulator is an emulation/simulation platform developed for the Contiki OS. It is an extensible Java-based simulator able to simulate the network, operating system, and instruction set. It is also able to emulate the execution of the exact same firmware that may be uploaded to physical nodes, instead of simulating it. Cooja allows developers to test their code and systems long before running it on the target hardware.

TOSSIM http://tinyos.stanford.edu/ [LL03]

TOSSIM is a wireless sensor network simulator that was built with the specific goal to simulate TinyOS devices. Since TinyOS is event-based, it is easily translated into a simulator engine with discrete-events, thus simplifying it and making it more effective. TOSSIM supports two programming interfaces (Python, C++), and has various levels of simulation, from hardware interrupts to high-level system events, such as packet arrivals.

iFogSim http://www.cloudbus.org/cloudsim/

iFogSim is a Fog Computing Simulator able to simulate edge devices, cloud data centers, and network links, and perform metrics evaluation on them. With these features, it allows investigation and comparison of resource management techniques based on QoS (Quality-of-Service) criteria (e.g. latency, network congestion).

MobIoTSim https://github.com/sed-szeged/MobIoTSim [PKSL16]

MobIoTSim is a mobile IoT device simulator, developed in Android, designed to help researchers learn IoT device handling without buying real sensors, and to test and demonstrate IoT applications utilizing multiple devices. This system can be connected to a gateway service in a cloud, such as IBM Bluemix Platform and Azure IoT Hub, to manage the simulated devices and to send back notifications by responding to critical sensor values. By using this tool, developers can examine the behavior of small IoT systems, and evaluate IoT cloud applications with a hand-held device.

IOTSim [ZGS+17]

IOTSim is a Cloud simulator built on top of the CloudSim system and designed to support the testing of IoT big data processing, resorting to a MapReduce approach. By inherently supporting big data systems, it facilitates the understanding and analysis of the impact and performance of IoT-based applications by researchers and commercial organizations.

DPWSim [HLC+14]

DPWSim is a simulation toolkit to support the prototyping and development of service-oriented and event-driven IoT applications. It aims to support the OASIS standard Devices Profile for Web Services (DPWS), which, although it enables the use of web services on smart and resource-constrained devices, reduces the scope of such a system to IoT devices that implement the referred device profile.

SimpleIoTSimulator https://www.smplsft.com/SimpleIoTSimulator.html

SimpleIoTSimulator is an IoT device simulator that can create test environments made up of thousands of sensors and gateways on a computer. It supports many of the common IoT protocols (e.g. CoAP) and is able to learn from data of recorded packet exchanges from real servers and sensors and model the behavior of its simulated devices from such data.

Atomiton IoT Simulator http://www.atomiton.com/

The Atomiton IoT Simulator, built atop Atomiton Stack (a proprietary operating environment for the Internet of Things), is a prototyping and testing framework able to simulate virtual sensors, actuators, and devices with unique behaviors. It allows for prototyping an IoT solution and testing its scalability by providing the ability to create boundary test cases, resorting to the simulation of thousands of devices and events such as network interruptions, device response delays, and peak load.

MBTAAS [ABF+16]

The Model-Based Testing as a Service (MBTAAS) allows the systematically test the IoT and data platforms. The approach resorts to a combination of model-based testing (MBT) techniques and service-oriented solutions. The solution has been tested on top of the FIWARE IoT-enabling platform. Further, the modularity of the solutions allows *integration testing* between different IoT platforms.

3.2 IoT Testing Solutions Comparison

An overview comparison of the available tools for testing IoT solutions is given in Table 3.1. Testing capabilities of each solution are analyzed by the observation of different variables.

In the first place, the tools are divided by the *IoT Layer* they focused on, as they are presented in Figure 2.1. Here it can be observed a relation between the layer and the testing variable related to. Edge layer tools, such as the `PlatformIO` and `ArduinoUnit`, typically focus on testing the code that runs on edge devices (e.g. Arduino). However, to test the edge layer the already available tools from embedded system testing can be helpful. Fog and Cloud tools are typically concerned about network or application testing, disregarding the low-level tests on code but testing at the System and Integration level.

By the analysis of the *Test Level* to which each tool is concerned about we notice that there are tools covering all levels, from *unit testing* until *acceptance testing*, at least in a partial way. We must note that although the tools enable one to test all the levels, they do not provide out-of-the-box functionalities to do so. Example of one of those is the `FIT IoT-LAB` testbed that provides a large-scale platform to test applications across the different layers, but requires development efforts in, for example, retrieve and manage data from that testing. In other cases, the tools provide only partial support for the testing functionalities, e.g., providing functionalities of collecting all network logs and responses but not providing direct insights on that information.

Some gaps appear in the solutions support of different languages and platforms. A vast part of the available tools focuses on a specific platform, language or standard, lacking the support for heterogeneity of the IoT field. Example of such tools are the `DPWSim` that focus on the Devices Profile for Web Services (DPWS) standard language and the `TOSSIM` simulator for the *TinyOS* compatible devices. Another problem appears from the large range of network communication protocols and IoT-enabling technologies (e.g. reference architectures) that are now appearing in the market without any kind of standardization, which leads to the lack of tools to test them in a platform-agnostic way. However, some have a large number of platforms supported or are open to any implementation requiring some extra development efforts.

Moving towards the different artifacts that need to be tested in the IoT landscape, the testing necessities are common to the highly distributed systems field. Firstly, and the artifact with more covered by the available solutions (e.g. `MAMMotH`, `iFogSim`), the network and communication variable. Secondly, with some available tools such as the `MobIoTSim`, the application level testing, in which the functionality, usability, and consistency can be tested within a real-world

scenario, disregarding the business logic behind them. Some solutions are also available for code testing for the edge devices such as the `PlatformIO`. However, it is easily noticeable that there is a lack of tools for testing certain artifacts such as security and privacy, regulatory testing, and firmware/software upgrade (e.g. out-of-the-box continuous integration functionalities).

In the security and privacy scope, there is work being pursued by the OWASP (Open Web Application Security Project) to “help manufacturers, developers, and consumers better understand the security issues associated with the Internet of Things, and to enable users in any context to make better security decisions when building, deploying, or assessing IoT technologies" [(OW17].

Testing environments are another distinguishable aspect within the available testing solutions. Most of the environments are purely virtual by means of emulation (e.g. virtual representation of an Arduino board) or simulation techniques (e.g. simulation of a smart city or smart house). However, some efforts have been done in the creation of physical testbeds like the `FIT IoT-LAB`. Also, some traditional software testing tools are available (for unit testing purposes) that most of the times rely on physical devices to conduct the testing.

Another relevant aspect is the stage of development of the solutions found and their openness. It is observable that most of the solutions have been presented in the literature, however, most of them are purely academic and there is no access to its source code or the software package. Comparatively, the solutions available to be used are scarce and most of them are closed-source, reducing the possibility of extending the tool functionalities or improving it by the means of extensions or plug-ins. Here it can also be noted that some tools are only available on remote test runners which can reduce the ability to test specific needs of certain solutions and raise privacy concerns.

3.3 Conclusion

In this chapter, the current state-of-the-art systems developed for testing IoT environments have been described and its features compared. These systems were analyzed in multiple categories: IoT layer, test level, test method, testing artifact, programming language, test environment, test runner, supported platforms, scope, and license. With this data, we were able to develop an overview of the different approaches, their usefulness, and which issues are not able to be tackled by them.

Next chapter, the findings from this analysis of the current state of the IoT testing environment will be used to project a new system that tackles some of the presented issues.

Tool	IoT Layer	Test Level	Test Method	Testing Artifact	Prog. Lang.	Test Environment	Test Runner	Sup. Platforms	Scope	License
PlatformIO	Edge	Unit	White-box	Code	C/C++, Arduino	Device	Local , Remote	15+	Market	Closed
IoTIFY	All	Any	White-box	N/A	N/A	Simulator	Remote	N/A	Market	Closed
FIT IoT-LAB	All	Any	Any	N/A	N/A	Physical Testbed	Local, Remote	6+	Academic, Market	Open
ArduinoUnit	Edge	Unit	White-box	Code	Arduino	Device	Local	Arduino	Academic, Market	Open
MAMMotH	All	Integration, System	Any	Network	N/A	Emulator	Local	N/A	Academic	N/A
Cooja	Edge	Integration	Black-box	Network	C	Emulator	Local	Contiki OS	Academic, Market	N/A
TOSSIM	Edge	Integration	Any	Application, Network	Python, C++	Simulator	Local	TinyOS	Academic	Open
SWE Simulator	Edge	System	Black-box	Application, Network	XML, Visual	Simulator	Local	SWE Standard	Academic	N/A
SimIoT	Fog	Integration, System	Black-box	Any	N/A	Simulator	Local	N/A	Academic	N/A
iFogSim	Edge, Fog	Integration, System	Grey-box	Network	Java	Simulator	Local	N/A	Academic	Open
MobIoTSim	Fog, Cloud	Integration, System	Grey-box	Application, Network	N/A	Simulator	Local	N/A	Academic	Open
IOTSim	Cloud	Integration	Any	Application	N/A	Simulator	N/A	N/A	Academic	N/A
DPWSim	Fog, Cloud	Integration, System	Any	Application	WSDL	Simulator	Local	DPWS	Academic	N/A
SimpleIoTSimulator	Edge, Fog	Integration, System	Any	Network	N/A	Simulator	Local	N/A	Market	Closed
Atomiton IoT Simulator	All	Any	Grey-box	N/A	N/A	Simulator	Remote	N/A	Market	Closed
MBTAAS	All	Any	Black-box	Model	OCL	Platform	N/A	N/A	Academic	N/A

Table 3.1: Overview comparison of the available tools on the IoT testing landscape. N/A symbolizes that there is no information available or not to have been found during research.

Chapter 4

Thesis Statement

In this section the problems with the current testing environments of the Internet of Things and the proposed solution for them. After that, relevant features of the solution are described as well as the means of validation of said features.

4.1 Current Issues

As presented in chapter 3, each of the analyzed frameworks tackles IoT testing with a different approach, providing useful features, but also some limitations. Due to these limitations, the following desirable features in an IoT simulation environment are not all available simultaneously in the analyzed systems:

- **Public interface**: the ability to use the simulated system state in an application decoupled from the simulation environment during its execution.
- **Local testing environment**: the ability to restrict the testing environment to the local premises desired by the system users. In some solutions, the devices must be connected to or provided in a remote environment not controlled by the user, leading to security and privacy concerns.
- **Coexistence of simulated and physical devices**: the ability to have an environment with simulated and physical devices simultaneously.
- **Extensibility**: the ability to implement new features, instead of relying on a closed environment that may not provided all the needed ones.

4.2 Main Goal

We propose a framework that is focused in filling one of the gaps left by those solutions: the coexistence of simulated and physical devices in the same environment.

This framework should be able to fulfill a list of use cases hereinafter presented. Some are based on capabilities already provided in the currently available frameworks, while others describe functionalities that could enhance such systems.

4.2.1 Use Case 1: prototype an IoT environment without any physical device

The ability to create a fully virtual environment enables the prototyping of IoT solutions before acquiring the needed devices. This use case is the base requirement of an IoT solution virtual prototype, and, as such, should be achievable by any IoT simulation environment.

4.2.2 Use Case 2: use physical sensors in a simulated environment

Though simulated sensor data can be used to test an IoT solution, it may not be able to provide a complete understanding on how the solution would behave after being physically implemented. As such, integrating data gathered directly from physical sensors in real time allows for a better prediction on its behavior.

4.2.3 Use Case 3: provide simulated sensor data to the physical environment

In contrast to Use Case 2, when an IoT solution is already physically implemented, it can be hard to study how the environment would react in anomalous conditions. In order to enable its study, simulated sensor data can be integrated in the environment to provide those anomalous conditions for testing.

4.2.4 Use Case 4: control physical actuators from a simulated environment

The IoT environments is not only populated by sensor devices, but also by actuator devices. As such, frameworks should be able to control them to properly test their behavior.

4.2.5 Use Case 5: Execute test batteries

Though many IoT environments are simple in design, some others are rather complex and extensive testing of those systems can not be done by people, due to location and duration constraints. An environment able to define different scenarios, by remotely controlling the devices and defining frequency of the events in the system, would simplify the testing process of said systems.

4.3 Research Questions

In accordance with the main goal and the previously presented use cases, these research questions have been established:

1. **How can simulated and physical devices be easily interchangeable?**: in order to provide a realistic simulation, the ability to change between a simulated and a physical device allows a more confident implementation of an IoT environment after its prototyping.

2. **How will simulated and physical devices interact with each other?**: in continuation of the previous question, the interchangeability of physical and simulated devices is dependent on a proper communication channel between them, otherwise the physical and simulated devices would not be able to coexist in the same environment.

4.4 Evaluation Strategy

The developed framework will be tested by replicating common use case scenarios of a smart space environment and validating the events produced by the framework against ones expected in those scenarios. These scenarios are based on the aforementioned use cases, allowing the verification of the fulfillment of said use cases.

4.5 Conclusions & Expected Contributions

The Internet of Things is a relatively new field of software development with many issues to be tackled. By developing an extensible framework centered around the design of IoT scenarios with coexistence of simulated and physical devices, we expect to contribute with a new perspective for the current IoT testing environment.

Chapter 5

Implementation

5.1 Solution Overview

As mentioned in Section 4.2, this solution wants to provide an environment for testing IoT scenarios. To achieve such goal, it requires the components pictured in the Figure 5.1. The simulation framework is responsible for simulating the evolution of an IoT environment. The events produced are then published to the MQTT broker and can be used by applications decoupled from the framework. The MQTT broker is not only the public gateway use for the simulation events, but is also the main communication hub between the simulated environment and the physical environment. The IoT devices present in the figure represent the physical devices that may be included in a IoT scenario defined with the framework.

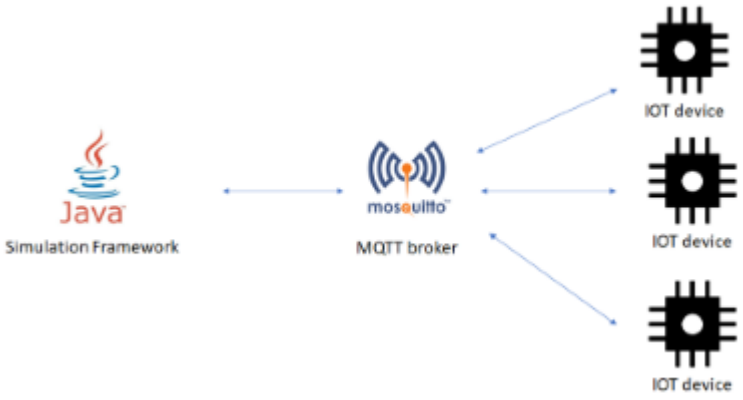

Figure 5.1: Architectural components of the IoT simulation system

5.2 Components Overview

In this section, the components referred in the previous section are described comprehensively, so that the functionalities and goals of each one are understood.

5.2.1 Simulation Framework

The simulation framework is a Java application that provides a simulation environment for the design of IoT scenarios. The simulation environment follows an event-based approach, though some tweaks to the canonical event-based simulation were implemented in order to better accommodate the simulated environment and its functionalities.

Next, the main components of the framework are described.

5.2.1.1 IoT Devices

In this system, IoT devices are represented as an identifier, a small optional description, a collection of properties, defined by an IoT device type, and a controller. Currently, the framework is ready to support the IoT device types defined in the proposed Web Thing API by Mozilla. This proposal features the following types:

- **thing**: generic type of device, in which no properties are mandatory.
- **onOffSwitch**: device with a boolean on/off property, such as a light switch.
- **multiLevelSwitch**: device with a boolean on/off property and a percent level property, such as a dimmer switch.
- **binarySensor**: generic type for sensor with a boolean on/off property, such as a movement sensor.
- **multiLevelSensor**: generic type for a sensor with a boolean on/off property and a percent level property.
- **smartPlug**: a sensor on a plug with a boolean on/off property, a value of its instantaneous power in Watts, as well as optional properties for voltage, current, frequency and level.
- **onOffLight**: a light that can be turned on or off.
- **dimmableLight**: a light with an on/off state and its luminosity level as a percentage.
- **onOffColorLight**: a light whose color can be set to a color specified as an RGB value.
- **onOffColorLight**: a light whose color can be set to a color specified as an RGB value and its intensity defined by a percentage.

In this framework, every device (simulated and physical) has a representation of its state present locally in the system. The device controller is responsible to keep the state of the device properly updated, by simply manipulating the collection of properties in the case of a simulated device or by communicating with and mirroring the state of a physical device.

5.2.1.2 Actors

Actors act as the main source of events in this system. Actors are able to interact with each other, schedule events and react to other events. In this simulation, two main actors are defined:

- **Client**: actors that interact with the IoT devices by watching their properties and requesting changes.
- **Device Controller**: actor responsible for an IoT device, able to change the internal status of the device and interpret requests sent by the Client actors.

5.2.1.3 Events

In a simulation environment, events explain the evolution of the system. For this solution, two relevant events where identified:

- **Property Value Change Events**: events emitted when the value of a device's property changes. They are defined by the time when such change occurred, the affected device, the affected property, and the new value. Listing 5.1 provides a JSON representation of the device "bedroom-switch" having its property labeled "on" change its value to false at time step 50, which translates in a switch being turned off.

```
{
    "eventType": "property-value-change",
    "triggeringTime": 50,
    "thingName": "bedroom-switch",
    "property": "on",
    "value": false
}
```

Listing 5.1: Event emitted when the value of an IoT device's property changes

- **Property Value Change Request Events**: events emitted when a Client requests a Device Controller for a change on one of the device's properties. They are defined by the time when such request occurred, the client who made the request, the targeted device, the targeted property and the new value. The listing 5.2 provides a JSON representation of a request from "bedroom-switch-light-linker" to the device "bedroom-light" in order to change its

property "on" to false at time step 51, which translates in a request to turn off the bedroom light.

```
{
    "eventType": "property-value-change-request",
    "triggeringTime": 51,
    "requesterIdentifier": "bedroom-switch-light-linker",
    "thingName": "bedroom-light",
    "property": "on",
    "value": false
}
```

Listing 5.2: Event emited for a request to change the value of an IoT device's property

5.2.1.4 Evolution of the Simulation

As described in Section 2.3, the virtual time of an event-based simulation is advanced according to the events occurring in the said environment. Since the simulation portrayed by the developed system revolves around events, a discrete-event simulation would provide the most number of processed events per execution time, by hopping from the current time step to the time step of the next event in the queue.

However, in order to more easily interact with physical devices and with other decoupled systems relying on this framework, the simulation is able to set a defined duration for each time step and no skipping of time steps occurs as in the previously described discrete event simulation implementation. This modification provides a relation between the simulation time and the real-time and is described in the Figure 5.2.

5.2.1.5 Implementation

The class diagram presented in Figure 5.3 provides an overview of the main components of the simulation system described in the previous sections. They are linked as such:

- IoT device: the IoT devices identification and properties are represented by the *Thing* class and corresponding *ThingType*. Their behavior is controlled by instances of the class *Thing-Controller*.
- Actors: actors have the base class *Actor* and the two described main actor types have a corresponding subclass:

  Client: client actors are implemented by the *Client* subclass

  Device Controller: device controller actors are implement by the *ThingController* subclass, further divided into Local or Physical controllers, depending whether the device is simulated or physical.

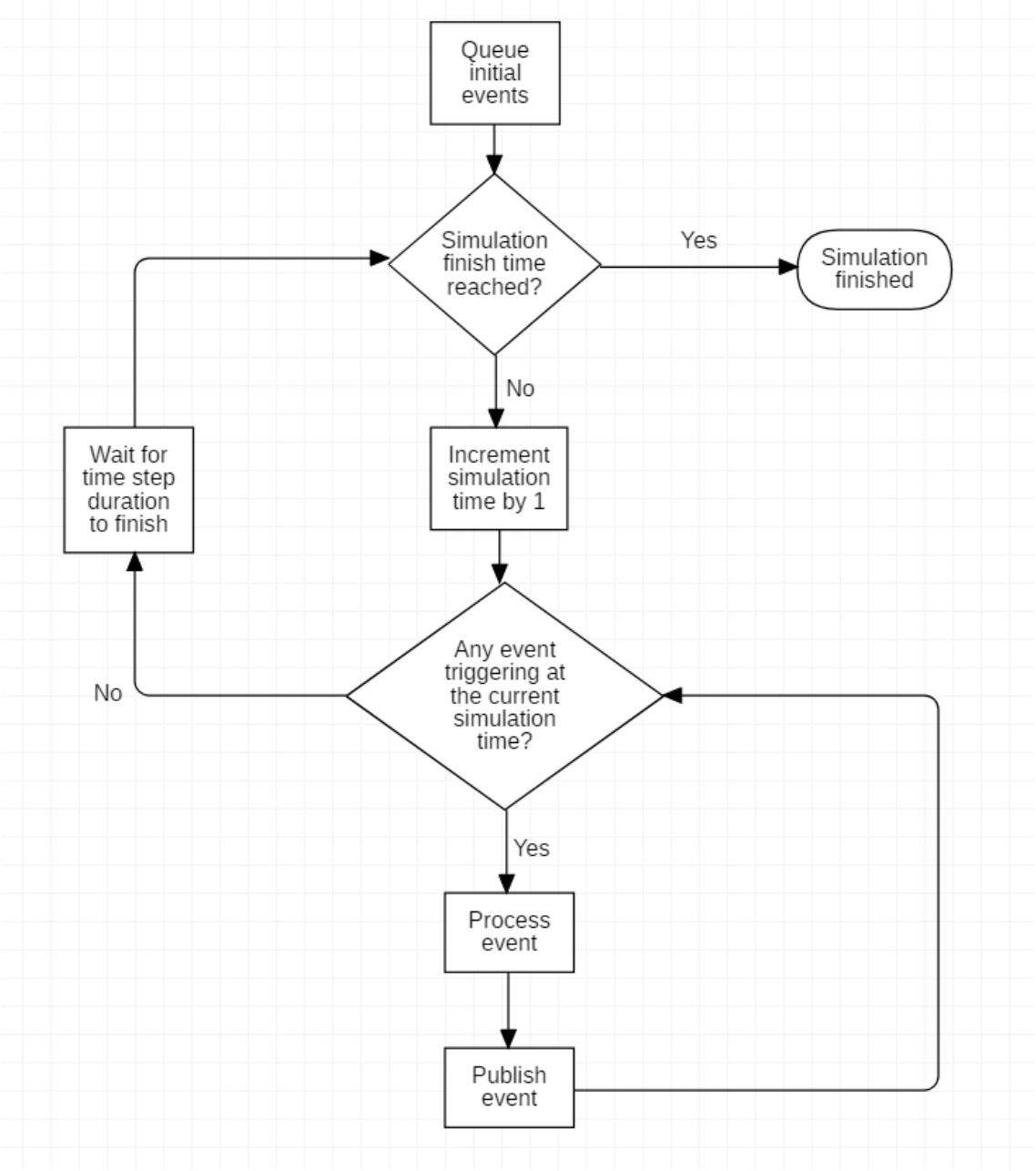

Figure 5.2: Modified event simulation execution flowchart

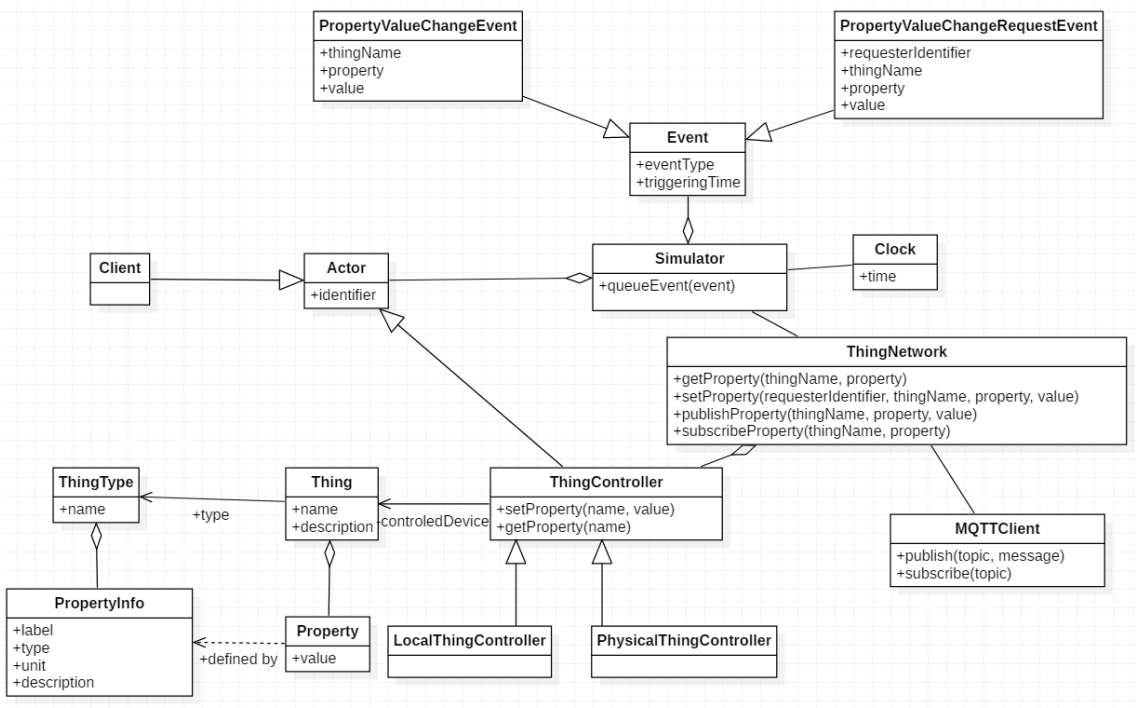

Figure 5.3: Class diagram for the simulation framework

- Events: each event inherits from the base class *Event* and both previously defined events have the following subclass implementation, each featuring the corresponding attributes:

  Property Value Change Event: *PropertyValueChangeEvent*

  Property Value Change Request Event: *PropertyValueChangeRequestEvent*

5.2.2 MQTT Broker

MQTT is a lightweight messaging protocol specially designed for low-resource communication between machines. It implements a publish/subscribe system and is one of the reigning communication hubs in Internet of Things environments. In this system, the MQTT broker provides not only the communication between the simulation and physical devices but also as an event bus where all the events occurring in the simulation are published for access of decoupled applications.

5.2.2.1 Event Bus

Though the event bus could be restricted inside the simulation environment of this system without affecting its execution, an external interface allows other systems to use the simulation events in a decoupled manner.

To achieve this objective, the events referred in 5.2.1.3 are published under the topic *simulation* during the simulation execution and can be accessed by subscribing to the said topic.

Some examples of systems that can make use of this feature are:

- **Persistent Data Storage**: storing the events emitted during a certain scenario, in order to later study its evolution.
- **Metrics and Alerts**: provide notifications about key features observed during the simulation.
- **Graphical Interfaces**: display the simulated environment in a more user-friendly method.

5.2.2.2 Communication Protocol

In an IoT environment, communication between devices is key for its proper functioning. As such, a communication protocol must be established and followed by the device's controllers and by the clients of each environment.

Using the MQTT Broker as the communication hub, the communication protocol implemented in this solution provides the interactions described in the following subsections.

The presented protocol is based on the Web Thing API proposed by Mozilla Foundation, whose draft is available in https://iot.mozilla.org/wot/. Since no definition for an MQTT environment was provided, an adaptation was created based on the REST and WebSocket definitions.

Thing Resource The Thing Resource provides a complete description of a device. It features its name, type, a small description and properties.

It is published by the device controller under the topic *things/{Device Name}* as a retained JSON message and can be retrieved by subscribing to the said topic.

```
{
  "name": "bedroom-light",
  "type": "onOffLight",
  "description": "Light in the bedroom",
  "properties": {
    "on": {
      "label": "On",
      "type": "boolean",
      "description": "Whether the light is turned on or off",
      "href": "/things/bedroom-light/properties/on"
    }
  }
}
```

Listing 5.3: Thing Resource example, published under the topic "/things/bedroom-light"

In the listing 5.3, an example of a Thing Resource for a device can be consulted. From the analysis of this Thing Resource, one can conclude that the device is a device of the *onOffLight* type labeled *bedroom-light* and is described as the "Light in the bedroom". This device has only one boolean property, which describes "Whether the light is turned on or off" and that can be consulted under the MQTT topic "/things/bedroom-light/properties/on".

Property Resource A Property Resource provides the current state of a property of an IoT device.

It is published by the device controller under the topic *things/{Device Name}/properties/{Property Name}* as a retained message and can be retrieved by subscribing to the said topic.

```
{
    "on": true
}
```

Listing 5.4: Property Resource example, published under the topic "things/bedroom-light/properties/on"

In the listing 5.4, an example of a Property Resource can be consulted. From the analysis of this Property Resource, one can conclude that the property *on* has the value *true* and it belongs to the device *bedroom-light*. However, in order to understand how to interpret this reported value, the information provided at the Thing Resource available under the topic "things/bedroom-light" must be consulted.

Property Value Change Request In order to allow client systems to request a change in the state of the IoT device, they must provide the device controller with the targeted property and the desired value.

Such information is provide by the client by publishing under the topic *things/{Target Device Name}/requests/{Client Identifier}*. The device controller is expected to subscribe to all the topics under the wildcard *things/{Device Name}/requests/+* in order to retrieve any request and properly react to it.

```
{
    "messageType": "setProperty",
    "data": {
      "on": true
    }
}
```

Listing 5.5: Property Value Change Request example, published under the topic "things/bedroom-light/requests/light-controller"

In the listing 5.5, an example of a Property Value Change Request can be consulted. From the analysis of this Property Value Change Request, one can conclude that the MQTT client identified as *light-controller* as issued a request to the device labeled *bedroom-light* in order to the change its property *on* to the value *true*.

5.2.3 IoT Device

Though out of the scope of the presented solution, these devices are needed for interaction with the physical world.

In order to include them in the system, their controller must be connected to the MQTT broker shared by the system, abiding to the communication protocol defined in the section 5.2.2.2, and its identification and status as a physical device must be included in the definition of the IoT scenario in the simulation framework.

5.3 Achieving Mixed-Reality Simulation

This framework whose implementation was described in the previous sections has one main objective: enable the interaction of simulated and physical devices during the execution of the simulation.

In the simulation framework, both kinds of devices are represented by the Device Controller actors, with corresponding subclasses for simulated and physical ones. These actors are responsible for the updates in the simulated and physical systems related to the simulation events described in Section 5.2.1.3. Though each kind of event is independent on the location of the device, the actions related to them vary according to it.

The two next subsections provide a description of the actions occurring in the system for each kind of event, dependent if the target device is simulated or not.

5.3.1 Simulated Devices

For simulated devices, a *LocalThingController* is responsible for the actions related to the aforementioned events.

A Property Value Change Event of a simulated device is emitted by the corresponding controller when it deems that a status change must be done. When the simulator processes this same event, the controller is tasked with updating the value of the property of the simulated device and then publish this new value in the *ThingNetwork*, which in turn publishes a message in que MQTT broker, at the corresponding topic according to the communication protocol defined in Section 5.2.2.2. Finally, the simulator broadcasts the event to all actors in the system. This procedure is depicted in Figure 5.4.

A Property Value Change Request Event for a simulated device can be emitted by any actor in the simulation. When the simulator processes this same event, the target device controller is tasked with analyzing the request and then publish this request in the *ThingNetwork*, which

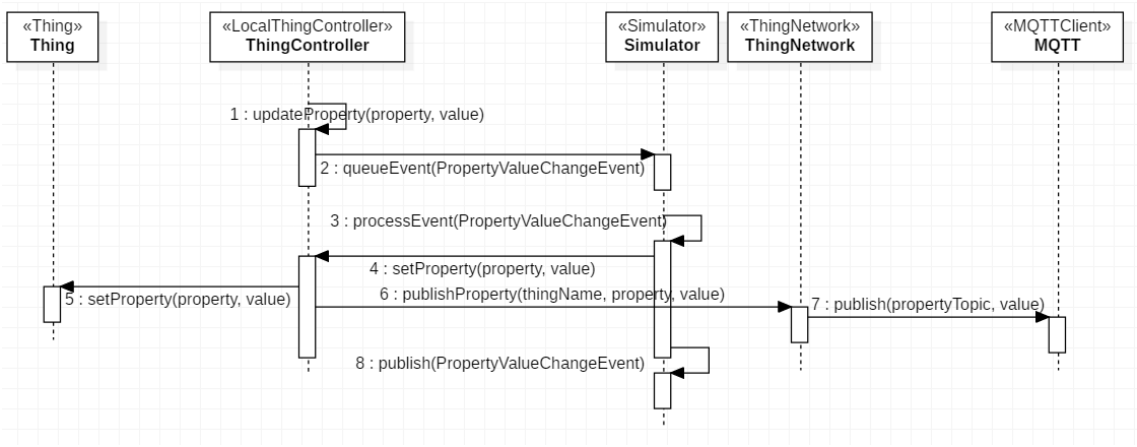

Figure 5.4: Property Value Change Event on a Simulated Device

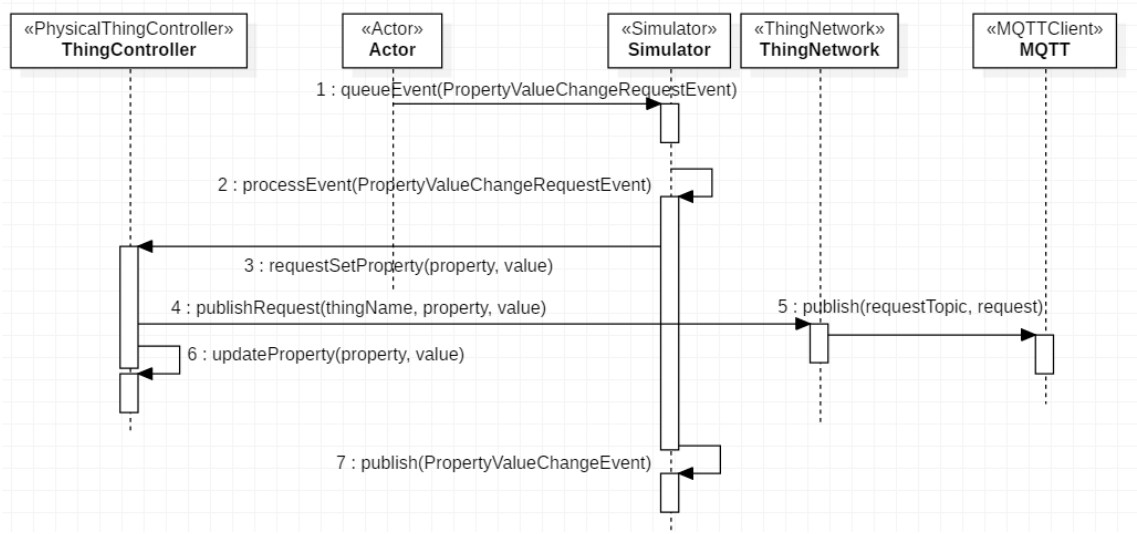

Figure 5.5: Property Value Change Request Event on a Simulated Device

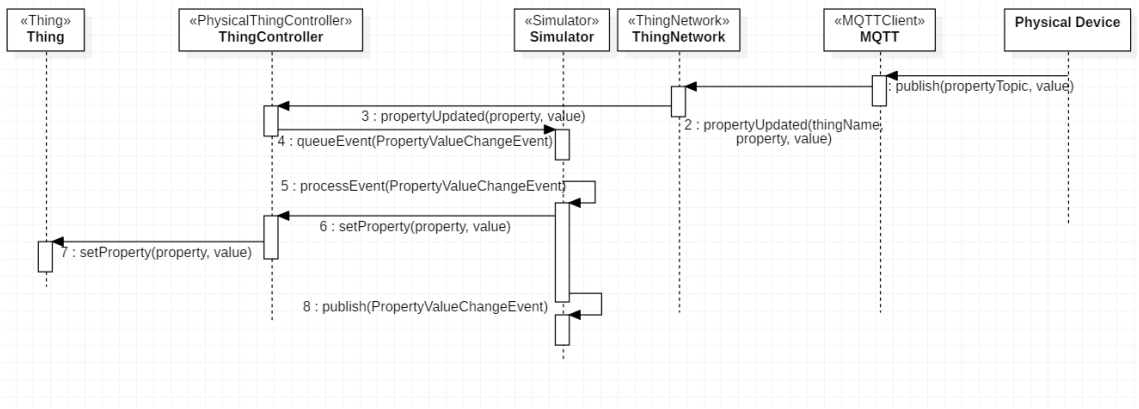

Figure 5.6: Property Value Change Event on a Physical Device

in turn publishes a message in the MQTT broker, at the corresponding topic according to the communication protocol defined in Section 5.2.2.2. Finally, the simulator broadcasts the event to all actors in the system. This procedure is depicted in Figure 5.5.

5.3.2 Physical Devices

For physical devices, a *PhysicalThingController* is responsible to watch the MQTT topics related to their corresponding device and inform the simulation about changes on the physical device.

A change on the value of a property of a physical device is detected by the corresponding controller when a new message is published by the physical device on the property MQTT topic. After a new value is detected, the controller must emit a Property Value Change Event. When the simulator processes this same event, the controller is only tasked with updating the value of the property of the local representation of the physical device. Finally, the simulator broadcasts the event to all actors in the system. This procedure is depicted in Figure 5.6.

A request for a change on the value of a property of a physical device is detected by the corresponding controller when a new message is published by the physical device on the request MQTT topic. After a new request is detected, the controller must emit a Property Value Change Request Event. When the simulator processes this same event, no actions of the controller are

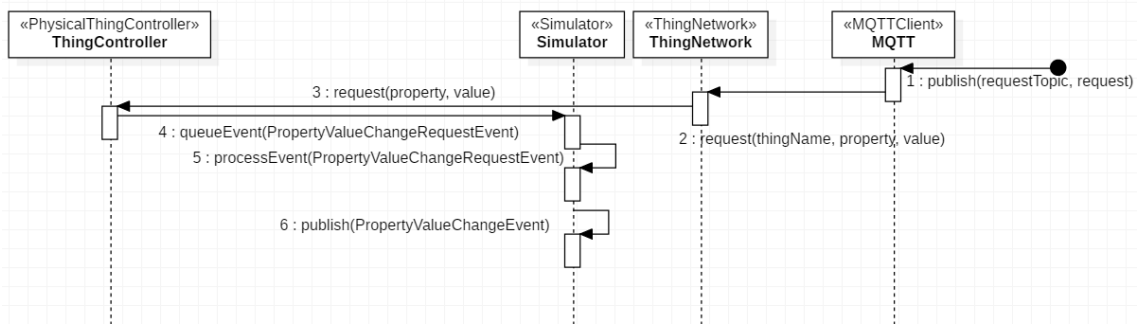

Figure 5.7: Property Value Change Request Event on a Physical Device

needed in the system, only the event broadcast to all the actors. This procedure is depicted in Figure 5.6.

5.4 Conclusions

The proposed solution for the issues presented in Chapter 4 is described in this chapter as a system composed by a simulation framework, a MQTT broker and physical IoT devices. Along with it, a MQTT communication protocol inspired by the Web Thing API proposed by Mozilla was designed in order to fulfill the interaction needs of the simulation system with the physical world. By leveraging this communication protocol, the simulation is able to reach a state of mixed-reality, with the simulation actors interacting with simulated and physical IoT devices.

Chapter 6

Evaluation

6.1 Use Case Scenarios

As defined in Section 4.4, Use Case Scenarios were designed in order to portray the fulfillment of the use cases defined in 4.2. In order to simplify the analysis of each Use Case Scenario, Section 6.1.1 provides an explanation of the parameters of the Use Case Scenarios presented in the following sections.

6.1.1 Use Case Scenarios Definition

The Use Case Scenarios describe potential Internet of Things' environments in which the use cases presented in Section 4.2 are reflected. They are defined by a collection of properties that are provided with the following format:

- Description: small description of the current scenario, depicting relevant events in the environment.
- Simulated devices: list of devices simulated by the framework present in the scenario. They are presented in the format:

  Device name: device type.
- Physical devices: list of non-simulated / physical devices present in the scenario. Similarly to the simulated devices, by are presented in the format:

  Device name: device type.
- Special device controller behavior: list of devices whose controllers implement additional behaviors than reporting its state and execute requests. The list follows the format:

  Device name: description of additional behavior.
- Client actors: list of non-device actors and description of corresponding behaviors in the format:

  Actor name: behavior description.

- Physical actions: list of actions needed to be executed in the physical environment.
- Expected behavior: definition of interactions expected in the environment, observable through the messages published in the MQTT broker.
- Time step duration: expected duration of a time step in execution time.
- Observed behavior: listing of messages published in the MQTT broker that represent the changes in the environment in the described scenario.
- Fulfilled use cases: list of the use cases defined in Section 4.2 that are observable in the described scenario.

6.1.2 Scenario 1: virtual environment

- Description: in this scenario, a small IoT environment with a single onOffLight device is simulated. In it, a client actor toggles the light in intervals of 20 seconds.
- Simulated devices:

  bedroom-light: OnOffLight.
- Physical devices: none.
- Client actors:

  light-flickerer: actor that toggles the bedroom-light every 20 time steps.
- Expected behavior: emission of a Property Value Change Request event followed by a Property Value Change event corresponding to the request change every 20 time steps.
- Observed behavior: Use Case Scenario 1 Log
- Fulfilled use cases: Use Case 1

6.1.3 Scenario 2: linking a physical switch to a physical light

- description: in this scenario, two physical devices are present: a switch and a light. An actor in the simulation environment must them match the state of the light to the state of the switch.
- Simulated devices: none.
- Physical devices:

  bedroom-light: OnOffLight.

  bedroom-switch: OnOffSwitch.

- Client actors:

  switch-light-linker: actor that watches the bedroom-switch for changes in its on/off state an updates the bedroom-light to the target state.

- Physical action: toggling the switch
- Expected behavior: batches of emissions of a Property Value Change on the bedroom-switch, followed by a Property Value Change Request on the bedroom-light and corresponding Property Value Change.
- Observed behavior: Use Case Scenario 2 Log
- Fulfilled use cases: Use Case 2, Use Case 4

6.1.4 Scenario 3: physical light updating own state based on simulated switch status

- Description: a physical light with an extended controller must update its status based on the status of a switch that is being simulated.
- Simulated devices:

  kitchen-switch: onOffSwitch

- Physical devices:

  kitchen-light: onOffLight

- Special device controller behavior:

  kitchen-light: it observes the kitchen-switch and mirrors its state.

- Client actors:

  switch-flickerer: it toggles the kitchen-switch every 20 time steps.

- Expected behavior: Property Value Changes on the kitchen-switch followed by Property Value Changes on the kitchen-light every 20 time steps.
- Observed behavior: Use Case Scenario 3 Log
- Fulfilled use cases: Use Case 3

6.1.5 Scenario 4: testing a 24-hour life cycle of self-controlled light based on a luminosity sensor at the simulation rate of 1h/min

- description: in this scenario, a light must be tested in accordance with the values provided during a 24h luminosity cycle. In order to speed up the process, a simulated device is used to provide the luminosity values in a shorter time-frame.

- Simulated devices:

  garden-luminosity: multiLevelSensor

- Physical devices:

  lounge-light: onOffLight

- Special device controller behavior:

  garden-luminosity: the controller updates the device state based on a 24-hour dataset of luminosity readings with 1 minute interval between each reading.

  lounge-light: keeps the light turned on if the reported luminosity levels are less than 40%, keeps it off otherwise.

- Time step duration: 60 time steps every 1 second of execution.

- Expected behavior: Property Value Change on the garden-luminosity level every 60 simulation time steps (corresponding to 1 second total in real time), followed by a Property Value Change on the lounge-light if it was on and the reported luminosity is below 40% or it if was off and it was reported that the luminosity level was equal or above 40%.

- Observed behavior: Use Case Scenario 4 Log

- Fulfilled use cases: Use Case 3, Use Case 5

6.2 Conclusions

The analysis of the previously describe Use Case Scenarios and corresponding message logs concludes that the developed framework is able to fulfill the desired objectives. These Use Case Scenarios were designed as a realization of the Use Cases presented in Section 4.2 and provide a validation of its usefulness in providing not only features expected of an IoT testing simulator, such as a fully virtual simulation, but also allows testers to create interactions between simulated and physical devices.

Chapter 7

Conclusions and Future Work

7.1 Main Problems

As it can be deduced from the multiple implementation of an IoT testing framework portrayed in chapter 3, every systems has problems of its own and this is no exception. During its development, these were the main problems identified:

- Availability of physical devices: the system does not take into account the communication errors between the simulation environment and the physical devices. Unless the system is in an environment that assures that no communication errors occur, these errors can lead to wrong assumptions about the scenario in test.

- Limited amount of supported device types: although the Web Thing API is a step forward in the communication between the heterogeneous devices of IoT environments, the amount of device types featured in it compared to the ones available is rather small. So, in order to support other device types, some code development is needed to easily accommodate the non-supported device types.

7.2 Conclusions

The presented solution was created in an attempt to tackle one of the many gaps in IoT testing systems: lack of systems were simulated and physical devices are able to coexist. This detected gap was further developed into specific use cases for the tool, which where reported in Section 4.2.

Thereafter, as described in Chapter 5, a system composed by a simulation framework, a MQTT broker and physical IoT devices was created, and the Web Thing API proposed by Mozilla inspired the design of a communication protocol to fulfill the interaction needs of the system.

In order to validate the developed system, Use Case Scenarios were created and exposed in Chapter 6. These Use Case Scenarios served the purpose of materializing the requirements specified in Chapter 4, allowing its fulfillment to be successfully assessed.

Though it is one more testing framework for the Internet of Things, we can ascertain that the ability to interact between the simulated and the physical environments provides a new perspective with which users can design and test their IoT solutions with more confidence on the end result.

7.3 Future Work

As an extensible framework and with the unceasing evolution of the IoT environment, there is always room for improvement in a testing framework for the Internet of Things. In accordance with the main problems, encountered during development and exposed in Section 7.1, and other issues of the IoT environment as a whole, some future improvements are here proposed:

- **Reinforce the support of the Mozilla Web Thing API**: though based on the Mozilla Web Thing API, the current communication protocol lacks some of its features, such as actions and events.
- **Detection of physical device properties**: in its current implementation, the system is not able to interpret the Thing Resource of a device in order to detect additional properties not defined in its type. With this limitation, the system is not able to analyze changes occurring in such properties.
- **Setup of scenarios through resource files**: in order to currently create a scenario, its specification must be coded into a Java Application with the simulation framework. A definition of a resource file format in which the devices, actors and other variables are defined would allow for a less cumbersome creation of multiple IoT scenarios.

References

[ABF+15] Cedric Adjih, Emmanuel Baccelli, Eric Fleury, Gaetan Harter, Nathalie Mitton, Thomas Noel, Roger Pissard-Gibollet, Frederic Saint-Marcel, Guillaume Schreiner, Julien Vandaele, et al. Fit iot-lab: A large scale open experimental iot testbed. In *IEEE 2nd World Forum on Internet of Things*, pages 459–464. IEEE, 2015.

[ABF+16] Abbas Ahmad, Fabrice Bouquet, Elizabeta Fourneret, Franck Le Gall, and Bruno Legeard. *Model-Based Testing as a Service for IoT Platforms*, pages 727–742. Springer International Publishing, Cham, 2016.

[BE15] B Bagula and ZENVILLE Erasmus. Iot emulation with cooja. In *ICTP-IoT Workshop*, 2015.

[Bei03] Boris Beizer. *Software testing techniques*. Dreamtech Press, 2003.

[BLC+11] Xiaoying Bai, Muyang Li, Bin Chen, Wei-Tek Tsai, and Jerry Gao. Cloud testing tools. In *Service Oriented System Engineering (SOSE), 2011 IEEE 6th International Symposium on*, pages 1–12. IEEE, 2011.

[CE11] L. Coetzee and J. Eksteen. The internet of things - promise for the future? an introduction. In *2011 IST-Africa Conference Proceedings*, pages 1–9, May 2011.

[Edw01] Stephen H Edwards. A framework for practical, automated black-box testing of component-based software. *Software Testing, Verification and Reliability*, 11(2):97–111, 2001.

[Gar17] Inc. Gartner. Gartner says 8.4 billion connected "things" will be in use in 2017, up 31 percent from 2016, 2017.

[GKN+11] Alexander Gluhak, Srdjan Krco, Michele Nati, Dennis Pfisterer, Nathalie Mitton, and Tahiry Razafindralambo. A survey on facilities for experimental internet of things research. *IEEE Communications Magazine*, 49(11), 2011.

[HLC+14] Son N. Han, Gyu Myoung Lee, Noel Crespi, Kyongwoo Heo, Nguyen Van Luong, Mihaela Brut, and Patrick Gatellier. DPWSim: A simulation toolkit for IoT applications using devices profile for web services. *2014 IEEE World Forum on Internet of Things, WF-IoT 2014*, pages 544–547, 2014.

[IEE90] IEEE. IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*, pages 1–84, Dec 1990.

[KK16] Ruslan Kirichek and Andrey Koucheryavy. *Internet of Things Laboratory Test Bed*, pages 485–494. Springer India, New Delhi, 2016.

REFERENCES

[Koo11] Philip Koopman. Embedded software testing, 2011.

[LJX+04] Wang Linzhang, Yuan Jiesong, Yu Xiaofeng, Hu Jun, Li Xuandong, and Zheng Guo. Generating test cases from UML activity diagram based on gray-box method. In *Software Engineering Conference, 2004. 11th Asia-Pacific*, pages 284–291. IEEE, 2004.

[LL03] Philip Levis and Nelson Lee. Tossim: A simulator for tinyos networks. *UC Berkeley, September*, 24, 2003.

[LODYJ12] Vilen Looga, Zhonghong Ou, Yang Deng, and Antti Yla-Jaaski. Mammoth: A massive-scale emulation platform for internet of things. In *Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on*, volume 3, pages 1235–1239. IEEE, 2012.

[LXZ15] Shancang Li, Li Da Xu, and Shanshan Zhao. The internet of things: a survey. *Information Systems Frontiers*, 17(2):243–259, Apr 2015.

[Ost02] Thomas Ostrand. White-Box Testing. *Encyclopedia of Software Engineering*, 2002.

[(OW17] Open Web Application Security Project (OWASP). Tester iot security guidance, 2017.

[PKSL16] T. Pflanzner, A. Kertesz, B. Spinnewyn, and S. Latre. MobIoTSim: Towards a mobile IoT device simulator. *Proceedings - 2016 4th International Conference on Future Internet of Things and Cloud Workshops, W-FiCloud 2016*, pages 21–27, 2016.

[RTS10] Leah Muthoni Riungu, Ossi Taipale, and Kari Smolander. Research issues for software testing in the cloud. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 557–564. IEEE, 2010.

[SBAM14] Stelios Sotiriadis, Nik Bessis, Eleana Asimakopoulou, and Navonil Mustafee. Towards simulating the internet of things. *IEEE 28th International Conference on Advanced Information Networking and Applications Workshops*, pages 444–448, 2014.

[TM17] Antero Taivalsaari and Tommi Mikkonen. A Roadmap to the Programmable World: Software Challenges in the IoT Era. *IEEE Software*, 34(1):72–80, 2017.

[WADX15] Andrew Whitmore, Anurag Agarwal, and Li Da Xu. The internet of things—a survey of topics and trends. *Information Systems Frontiers*, 17(2):261–274, 2015.

[ZCW+14] Z. K. Zhang, M. C. Y. Cho, C. W. Wang, C. W. Hsu, C. K. Chen, and S. Shieh. Iot security: Ongoing challenges and research opportunities. In *2014 IEEE 7th International Conference on Service-Oriented Computing and Applications*, pages 230–234, Nov 2014.

[ZGS+17] Xuezhi Zeng, Saurabh Kumar Garg, Peter Strazdins, Prem Prakash Jayaraman, Dimitrios Georgakopoulos, and Rajiv Ranjan. IOTSim: A simulator for analysing IoT applications. *Journal of Systems Architecture*, 72:93–107, 2017.

[ZKP00] Bernard P. Zeigler, Tag Gon Kim, and Herbert Praehofer. *Theory of Modeling and Simulation*. Academic Press, Inc., 2000.

Appendix A

Use Case Scenarios Data

A.1 Scenario 1: virtual environment

```
[2018-06-07 16:47:21] things/bedroom-light:
{
  "name": "bedroom-light",
  "type": "onOffLight",
  "description": "Light in the bedroom",
  "properties": {
    "on": {
      "label": "On",
      "type": "boolean",
      "description": "Whether the light is turned on or off",
      "href": "/things/bedroom-light/properties/on"
    }
  }
}

[2018-06-07 16:47:21] things/bedroom-light/properties/on:
{
    "on": false
}

[2018-06-07 16:47:41] things/bedroom-light/requests/light-flickerer:
{
  "messageType": "setProperty",
  "data": {
    "on": true
```

```
  }
}

[2018-06-07 16:47:41] simulation:
{
  "eventType": "property-value-change-request",
  "triggeringTime": 20,
  "requesterIdentifier": "light-flickerer",
  "thingName": "bedroom-light",
  "property": "on",
  "value": true
}

[2018-06-07 16:47:41] things/bedroom-light/properties/on:
{
  "on": true
}

[2018-06-07 16:47:41] simulation:
{
    "eventType": "property-value-change",
    "triggeringTime": 20,
    "thingName": "bedroom-light",
    "property": "on",
    "value": true
}

[2018-06-07 16:48:01] things/bedroom-light/requests/light-flickerer:
{
  "messageType": "setProperty",
  "data": {
    "on": false
  }
}

[2018-06-07 16:48:01] simulation:
{
  "eventType": "property-value-change-request",
  "triggeringTime": 40,
  "requesterIdentifier": "light-flickerer",
```

```
  "thingName": "bedroom-light",
  "property": "on",
  "value": false
}

[2018-06-07 16:48:01] things/bedroom-light/properties/on:
{
  "on": false
}

[2018-06-07 16:48:01] simulation:
{
  "eventType": "property-value-change",
  "triggeringTime": 40,
  "thingName": "bedroom-light",
  "property": "on",
  "value": false
}

[2018-06-07 16:48:21] things/bedroom-light/requests/light-flickerer:
{
  "messageType": "setProperty",
  "data": {
    "on": true
  }
}

[2018-06-07 16:48:21] simulation:
{
  "eventType": "property-value-change-request",
  "triggeringTime": 60,
  "requesterIdentifier": "light-flickerer",
  "thingName": "bedroom-light",
  "property": "on",
  "value": true
}

[2018-06-07 16:48:21] things/bedroom-light/properties/on:
{
  "on": true
```

```
}

[2018-06-07 16:48:21] simulation:
{
    "eventType": "property-value-change",
    "triggeringTime": 60,
    "thingName": "bedroom-light",
    "property": "on",
    "value": true
}

[2018-06-07 16:48:41] things/bedroom-light/requests/light-flickerer:
{
  "messageType": "setProperty",
  "data": {
    "on": false
  }
}

[2018-06-07 16:48:41] simulation:
{
  "eventType": "property-value-change-request",
  "triggeringTime": 80,
  "requesterIdentifier": "light-flickerer",
  "thingName": "bedroom-light",
  "property": "on",
  "value": false
}

[2018-06-07 16:48:41] things/bedroom-light/properties/on:
{
  "on": false
}

[2018-06-07 16:48:41] simulation:
{
  "eventType": "property-value-change",
  "triggeringTime": 80,
  "thingName": "bedroom-light",
  "property": "on",
```

```
  "value": false
}
```

Listing A.1: Scenario 1 MQTT messages log

A.2 Scenario 2: linking a physical switch to a physical light

```
[2018-06-07 16:50:17] things/bedroom-light:
{
  "name": "bedroom-light",
  "type": "onOffLight",
  "description": "Light in the bedroom",
  "properties": {
    "on": {
      "label": "On",
      "type": "boolean",
      "description": "Whether the light is turned on or off",
      "href": "/things/bedroom-light/properties/on"
    }
  }
}

[2018-06-07 16:50:17] things/bedroom-light/properties/on:
{
    "on": false
}

[2018-06-07 16:50:17] things/bedroom-switch:
{
  "name": "bedroom-switch",
  "type": "onOffSwitch",
  "description": "Switch in the bedroom",
  "properties": {
    "on": {
      "label": "On",
      "type": "boolean",
      "description": "Whether the switch is turned on or off",
      "href": "/things/bedroom-switch/properties/on"
```

```
    }
  }
}

[2018-06-07 16:50:17] things/bedroom-switch/properties/on:
{
    "on": false
}

[2018-06-07 16:50:17] things/bedroom-switch/properties/on:
{
  "on": true
}

[2018-06-07 10:50:32] simulation:
{
    "eventType": "property-value-change",
    "triggeringTime": 15,
    "thingName": "bedroom-switch",
    "property": "on",
    "value": true
}

[2018-06-07 16:50:32] things/bedroom-light/requests/switch-light-
   linker:
{
  "messageType": "setProperty",
  "data": {
    "on": true
  }
}

[2018-06-07 16:50:32] simulation:
{
  "eventType": "property-value-change-request",
  "triggeringTime": 15,
  "requesterIdentifier": "switch-light-linker",
  "thingName": "bedroom-light",
  "property": "on",
  "value": true
```

```
}

[2018-06-07 16:50:32] things/bedroom-light/properties/on:
{
  "on": true
}

[2018-06-07 16:50:32] simulation:
{
  "eventType": "property-value-change",
  "triggeringTime": 15,
  "thingName": "bedroom-light",
  "property": "on",
  "value": true
}

[2018-06-07 16:51:11] things/bedroom-switch/properties/on:
{
  "on": false
}

[2018-06-07 16:51:11] simulation:
{
    "eventType": "property-value-change",
    "triggeringTime": 54,
    "thingName": "bedroom-switch",
    "property": "on",
    "value": false
}

[2018-06-07 16:51:11] things/bedroom-light/requests/switch-light-
   linker:
{
  "messageType": "setProperty",
  "data": {
    "on": false
  }
}

[2018-06-07 16:51:11] simulation:
```

```
{
  "eventType": "property-value-change-request",
  "triggeringTime": 54,
  "requesterIdentifier": "switch-light-linker",
  "thingName": "bedroom-light",
  "property": "on",
  "value": false
}

[2018-06-07 16:51:11] things/bedroom-light/properties/on:
{
  "on": false
}

[2018-06-07 16:51:11] simulation:
{
  "eventType": "property-value-change",
  "triggeringTime": 54,
  "thingName": "bedroom-light",
  "property": "on",
  "value": false
}

[2018-06-07 16:51:43] things/bedroom-switch/properties/on:
{
  "on": true
}

[2018-06-07 16:51:43] simulation:
{
    "eventType": "property-value-change",
    "triggeringTime": 86,
    "thingName": "bedroom-switch",
    "property": "on",
    "value": true
}

[2018-06-07 16:51:43] things/bedroom-light/requests/switch-light-
   linker:
{
```

```
  "messageType": "setProperty",
  "data": {
    "on": true
  }
}

[2018-06-07 16:51:43] simulation:
{
  "eventType": "property-value-change-request",
  "triggeringTime": 86,
  "requesterIdentifier": "switch-light-linker",
  "thingName": "bedroom-light",
  "property": "on",
  "value": true
}

[2018-06-07 16:51:43] things/bedroom-light/properties/on:
{
  "on": true
}

[2018-06-07 16:51:43] simulation:
{
  "eventType": "property-value-change",
  "triggeringTime": 86,
  "thingName": "bedroom-light",
  "property": "on",
  "value": true
}
```

Listing A.2: Scenario 2 MQTT messages log

A.3 Scenario 3: physical light updating own state based on simulated switch status

```
[2018-06-07 17:01:34] things/bedroom-light:
{
  "name": "bedroom-light",
```

```
  "type": "onOffLight",
  "description": "Light in the bedroom",
  "properties": {
    "on": {
      "label": "On",
      "type": "boolean",
      "description": "Whether the light is turned on or off",
      "href": "/things/bedroom-light/properties/on"
    }
  }
}

[2018-06-07 17:01:34] things/bedroom-light/properties/on:
{
    "on": false
}

[2018-06-07 17:01:34] things/bedroom-switch:
{
  "name": "bedroom-switch",
  "type": "onOffSwitch",
  "description": "Switch in the bedroom",
  "properties": {
    "on": {
      "label": "On",
      "type": "boolean",
      "description": "Whether the switch is turned on or off",
      "href": "/things/bedroom-switch/properties/on"
    }
  }
}

[2018-06-07 17:01:34] things/bedroom-switch/properties/on:
{
    "on": false
}

[2018-06-07 17:01:51] things/bedroom-switch/properties/on:
{
  "on": true
```

```
}

[2018-06-07 17:01:51] simulation:
{
    "eventType": "property-value-change",
    "triggeringTime": 17,
    "thingName": "bedroom-switch",
    "property": "on",
    "value": true
}

[2018-06-07 17:01:51] things/bedroom-light/properties/on:
{
  "on": true
}

[2018-06-07 17:01:51] simulation:
{
  "eventType": "property-value-change",
  "triggeringTime": 17,
  "thingName": "bedroom-light",
  "property": "on",
  "value": true
}

[2018-06-07 17:02:17] things/bedroom-switch/properties/on:
{
  "on": false
}

[2018-06-07 17:02:17] simulation:
{
    "eventType": "property-value-change",
    "triggeringTime": 43,
    "thingName": "bedroom-switch",
    "property": "on",
    "value": false
}

[2018-06-07 17:02:17] things/bedroom-light/properties/on:
```

```
{
  "on": false
}

[2018-06-07 17:02:17] simulation:
{
  "eventType": "property-value-change",
  "triggeringTime": 43,
  "thingName": "bedroom-light",
  "property": "on",
  "value": false
}

[2018-06-07 17:02:48] things/bedroom-switch/properties/on:
{
  "on": true
}

[2018-06-07 17:02:48] simulation:
{
    "eventType": "property-value-change",
    "triggeringTime": 74,
    "thingName": "bedroom-switch",
    "property": "on",
    "value": true
}

[2018-06-07 17:02:48] things/bedroom-light/properties/on:
{
  "on": true
}

[2018-06-07 17:02:48] simulation:
{
  "eventType": "property-value-change",
  "triggeringTime": 74,
  "thingName": "bedroom-light",
  "property": "on",
  "value": true
}
```

Listing A.3: Scenario 3 MQTT messages log

A.4 Scenario 4: testing a 24h lifecycle of self-controlled light based on a luminosity sensor at the rate of 1h/min

```
[2018-06-07 17:20:21] things/garden-luminosity:
{
  "name": "garden-luminosity",
  "type": "multiLevelSensor",
  "description": "Luminosity sensor in the garden",
  "properties": {
    "on": {
      "label": "On",
      "type": "boolean",
      "description": "Whether the sensor is turned on or off",
      "href": "/things/garden-luminosity/properties/on"
    },
    "level": {
      "label": "Luminosity",
      "type": "number",
      "unit": "percent",
      "description": "Luminosity level",
      "href": "/things/garden-luminosity/properties/level"
    }
  }
}

[2018-06-07 17:20:21] things/garden-luminosity/properties/on:
{
    "on": true
}

[2018-06-07 17:20:21] things/garden-luminosity/properties/level:
{
    "level": 100
}
```

```
[2018-06-07 17:20:21] things/lounge-light:
{
  "name": "lounge-light",
  "type": "onOffLight",
  "description": "Light in the lounge",
  "properties": {
    "on": {
      "label": "On",
      "type": "boolean",
      "description": "Whether the light is turned on or off",
      "href": "/things/lounge-light/properties/on"
    }
  }
}

[2018-06-07 17:20:21] things/lounge-light/properties/on:
{
    "on": false
}

[2018-06-07 17:20:21] things/garden-luminosity/properties/level:
{
  "level": 97
}

[2018-06-07 17:20:21] simulation:
{
    "eventType": "property-value-change",
    "triggeringTime": 0,
    "thingName": "garden-luminosity",
    "property": "level",
    "value": 97
}

... messages skipped

[2018-06-07 17:26:58] things/garden-luminosity/properties/level:
{
  "level": 40
```

```
}

[2018-06-07 17:26:58] simulation:
{
    "eventType": "property-value-change",
    "triggeringTime": 23820,
    "thingName": "garden-luminosity",
    "property": "level",
    "value": 40
}

[2018-06-07 17:26:59] things/garden-luminosity/properties/level:
{
  "level": 40
}

[2018-06-07 17:26:59] simulation:
{
    "eventType": "property-value-change",
    "triggeringTime": 23880,
    "thingName": "garden-luminosity",
    "property": "level",
    "value": 40
}

[2018-06-07 17:27:00] things/garden-luminosity/properties/level:
{
  "level": 39
}

[2018-06-07 17:27:00] simulation:
{
    "eventType": "property-value-change",
    "triggeringTime": 23940,
    "thingName": "garden-luminosity",
    "property": "level",
    "value": 39
}

[2018-06-07 17:27:00] things/lounge-light/properties/on:
```

```
{
  "on": true
}

[2018-06-07 17:27:00] simulation:
{
  "eventType": "property-value-change",
  "triggeringTime": 23941,
  "thingName": "lounge-light",
  "property": "on",
  "value": true
}

[2018-06-07 17:27:01] things/garden-luminosity/properties/level:
{
  "level": 39
}

[2018-06-07 17:27:01] simulation:
{
    "eventType": "property-value-change",
    "triggeringTime": 24000,
    "thingName": "garden-luminosity",
    "property": "level",
    "value": 39
}

[2018-06-07 17:27:02] things/garden-luminosity/properties/level:
{
  "level": 39
}

[2018-06-07 17:27:02] simulation:
{
    "eventType": "property-value-change",
    "triggeringTime": 24000,
    "thingName": "garden-luminosity",
    "property": "level",
    "value": 39
}
```

```
... messages skipped

[2018-06-07 17:39:43] things/garden-luminosity/properties/level:
{
  "level": 39
}

[2018-06-07 17:39:43] simulation:
{
    "eventType": "property-value-change",
    "triggeringTime": 69720,
    "thingName": "garden-luminosity",
    "property": "level",
    "value": 39
}

[2018-06-07 17:39:44] things/garden-luminosity/properties/level:
{
  "level": 39
}

[2018-06-07 17:39:44] simulation:
{
    "eventType": "property-value-change",
    "triggeringTime": 69780,
    "thingName": "garden-luminosity",
    "property": "level",
    "value": 39
}

[2018-06-07 17:39:45] things/garden-luminosity/properties/level:
{
  "level": 40
}

[2018-06-07 17:39:45] simulation:
{
    "eventType": "property-value-change",
    "triggeringTime": 69840,
```

```
    "thingName": "garden-luminosity",
    "property": "level",
    "value": 40
}

[2018-06-07 17:39:45] things/lounge-light/properties/on:
{
  "on": false
}

[2018-06-07 17:39:45] simulation:
{
  "eventType": "property-value-change",
  "triggeringTime": 69840,
  "thingName": "lounge-light",
  "property": "on",
  "value": false
}

[2018-06-07 17:39:46] things/garden-luminosity/properties/level:
{
  "level": 40
}

[2018-06-07 17:39:46] simulation:
{
    "eventType": "property-value-change",
    "triggeringTime": 69900,
    "thingName": "garden-luminosity",
    "property": "level",
    "value": 40
}

[2018-06-07 17:39:47] things/garden-luminosity/properties/level:
{
  "level": 41
}

[2018-06-07 17:39:47] simulation:
{
```

```
    "eventType": "property-value-change",
    "triggeringTime": 69960,
    "thingName": "garden-luminosity",
    "property": "level",
    "value": 41
}

... messages skipped


[2018-06-07 17:44:20] things/garden-luminosity/properties/level:
{
  "level": 98
}

[2018-06-07 17:44:20] simulation:
{
    "eventType": "property-value-change",
    "triggeringTime": 86340,
    "thingName": "garden-luminosity",
    "property": "level",
    "value": 98
}
```

Listing A.4: Scenario 4 MQTT messages log